

```

1  /*
2  UTMconversion.h - Library to convert in UTM coordinates.
3  TinyTrackGPS v0.7
4
5  Copyright © 2019-2021 Francisco Rafael Reyes Carmona.
6  All rights reserved.
7
8  rafael.reyes.carmona@gmail.com
9
10 This file is part of TinyTrackGPS.
11
12 TinyTrackGPS is free software: you can redistribute it and/or modify
13 it under the terms of the GNU General Public License as published by
14 the Free Software Foundation, either version 3 of the License, or
15 (at your option) any later version.
16
17 TinyTrackGPS is distributed in the hope that it will be useful,
18 but WITHOUT ANY WARRANTY; without even the implied warranty of
19 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
20 GNU General Public License for more details.
21
22 You should have received a copy of the GNU General Public License
23 along with TinyTrackGPS. If not, see <https://www.gnu.org/licenses/>.
24 */
25
26 #if ARDUINO >= 100
27   #include "Arduino.h"
28 #else
29   #include "WProgram.h"
30 #endif
31
32 #ifndef UTMconversion_h
33 #define UTMconversion_h
34
35 class GPS_UTM {
36 private:
37   int _h;
38   char _letter;
39   long _x;
40   long _y;
41
42 public:
43   GPS_UTM(){
44     _h = 0;
45     _letter = 'Z';
46     _x = 0L;
47     _y = 0L;
48   };
49
50   void UTM(double lati, double longi) {
51     /*
52     * Transformación de las coordenadas geográficas a UTM
53     */
54
55     /*// Sobre la geometría del delipsoide WGS84
56     double a = 6378137.0;
57     double b = 6356752.3142;
58
59     float e1 = sqrt((a*a) - (b*b))/a; ///< Excentricidad.
60     double e = sqrt(sq(a) - sq(b))/b; ///< Segunda excentricidad.

```

```

61     double e2 = sq(e); ///< al cuadrado. Usaremos esta directamente.
62     double c = sq(a) / b; ///< Radio Polar de Curvatura.
63 */
64
65 // Se realiza las declaraciones para agilizar el calculo a UTM de X e Y.
66 double e2 = 673949675659e-14; ///< Segunda excentricidad al cuadrado.
67 double e2_2 = 336974837829e-14; // (e2 / 2.0)
68 double c = 639959362580397e-8; ///< Radio Polar de Curvatura.
69 double PI_180 = 1745329251994e-14; // (PI / 180.0)
70 double alf = 505462256744e-14; // (0.75 * e2)
71 double bet = 425820155e-13; // ((5.0 / 3.0) * alf * alf)
72 double gam = 16740579e-14; // ((35.0 / 27.0) * alf * alf * alf)
73
74     ///< Sobre la longitud y latitud. Conversión de grados decimales a radianes.
75
76     /*
77     * Cálculo del signo de la longitud:
78     *     - Si la longitud está referida al Oeste del meridiano de Greenwich,
79     *     entonces la longitud es negativa (-).
80     *     - Si la longitud está referida al Este del meridiano de Greenwich,
81     *     entonces la longitud es positiva (+).
82     */
83
84     double latRad = lati * PI_180; ///< Latitud en Radianes.
85     double lonRad = longi * PI_180; ///< Longitud en Radianes.
86
87     ///< Sobre el huso.
88     ///< float huso = ((longi + 180.0) / 6.0) + 1.0; ///< Nos interesa quedarnos solo
con la parte entera.
89     ///< _h = (int)huso;
90     _h = (int)((longi + 180.0) / 6.0) + 1;
91
92     ///< Handle special case of west coast of Norway
93     if ( lati >= 56.0 && lati < 64.0 && longi >= 3.0 && longi < 12.0 ) {
94         _h = 32;
95     }
96
97     ///< Special zones for Svalbard
98     if ( lati >= 72.0 && lati < 84.0 ) {
99         if ( longi >= 0.0 && longi < 9.0 ) _h = 31;
100         else if ( longi >= 9.0 && longi < 21.0 ) _h = 33;
101         else if ( longi >= 21.0 && longi < 33.0 ) _h = 35;
102         else if ( longi >= 33.0 && longi < 42.0 ) _h = 37;
103     }
104
105     int landa0 = _h * 6 - 183; ///< Cálculo del meridiano central del huso en
grados.
106     double Dlanda = lonRad - (landa0 * PI_180); ///< Desplazamiento del punto a
calcular con respecto al meridiano central del huso.
107
108     if ((84 >= lati) && (lati >= 72))
109         _letter = 'X';
110     else if ((72 > lati) && (lati >= 64))
111         _letter = 'W';
112     else if ((64 > lati) && (lati >= 56))
113         _letter = 'V';
114     else if ((56 > lati) && (lati >= 48))
115         _letter = 'U';
116     else if ((48 > lati) && (lati >= 40))
117         _letter = 'T';

```

```

118     else if ((40 > lati) && (lati >= 32))
119         _letter = 'S';
120     else if ((32 > lati) && (lati >= 24))
121         _letter = 'R';
122     else if ((24 > lati) && (lati >= 16))
123         _letter = 'Q';
124     else if ((16 > lati) && (lati >= 8))
125         _letter = 'P';
126     else if (( 8 > lati) && (lati >= 0))
127         _letter = 'N';
128     else if (( 0 > lati) && (lati >= -8))
129         _letter = 'M';
130     else if ((-8 > lati) && (lati >= -16))
131         _letter = 'L';
132     else if ((-16 > lati) && (lati >= -24))
133         _letter = 'K';
134     else if ((-24 > lati) && (lati >= -32))
135         _letter = 'J';
136     else if ((-32 > lati) && (lati >= -40))
137         _letter = 'H';
138     else if ((-40 > lati) && (lati >= -48))
139         _letter = 'G';
140     else if ((-48 > lati) && (lati >= -56))
141         _letter = 'F';
142     else if ((-56 > lati) && (lati >= -64))
143         _letter = 'E';
144     else if ((-64 > lati) && (lati >= -72))
145         _letter = 'D';
146     else if ((-72 > lati) && (lati >= -80))
147         _letter = 'C';
148     else
149         _letter = 'Z'; // This is here as an error flag to show that the latitude is
outside the UTM limits
150
151
152     /*!
153      * Ecuaciones de Coticchia-Surace para el paso de Geográficas a UTM (Problema
directo);
154      */
155
156     /// Cálculo de Parámetros.
157     double coslatRad = cos(latRad);
158     double coslatRad2 = sq(coslatRad);
159
160     double A = coslatRad * sin(Dlanda);
161     double xi = 0.5 * log((1 + A) / (1 - A));
162     double n = atan(tan(latRad) / cos(Dlanda)) - latRad;
163     double v = (c / sqrt(1 + e2 * coslatRad2)) * 0.9996;
164     double z = e2_2 * sq(xi) * coslatRad2;
165     double A1 = sin(2 * latRad);
166     double A2 = A1 * coslatRad2;
167     double J2 = latRad + (A1 / 2.0);
168     double J4 = (3.0 * J2 + A2) / 4.0;
169     double J6 = (5.0 * J4 + A2 * coslatRad2) / 3.0;
170     double Bfi = 0.9996 * c * (latRad - alf * J2 + bet * J4 - gam * J6);
171
172     /*!
173      * Cálculo final de coordenadas UTM
174      */
175     /*

```

```

176     Serial.println (" Las coordenadas GPS que se van a transformar son: ");
177     Serial.print (" Latitud: "); Serial.println (lati,6);
178     Serial.print (" Longitud: "); Serial.println (longi,6);
179
180     Serial.println (" Coordenadas UTM actuales: ");
181     Serial.print("UTM: "); Serial.print(_h); Serial.print("
");Serial.println(_letter);
182     */
183     _x = round(xi * v * (1 + (z / 3.0)) + 500000);
184     /*!< 500.000 es el retranqueo que se realiza en cada huso sobre el origen de
185     coordenadas en el eje X con el objeto de que no existan coordenadas negativas.
186     */
187     _y = round(n * v * (1 + z) + Bfi);
188     if (lati < 0.0) _y += 10000000;
189     /*!< En el caso de latitudes al sur del ecuador, se sumará al valor de Y
190     10.000.000
191     para evitar coordenadas negativas. */
192     /*
193     Serial.print (" X = "); Serial.print (_x); Serial.println (" (m)");
194     Serial.print (" Y = "); Serial.print (_y); Serial.println (" (m)");
195     */
196     };
197
198     int zone(){
199         return _h;
200     };
201
202     char band(){
203         return _letter;
204     };
205
206     long X(){
207         return _x;
208     };
209
210     long Y(){
211         return _y;
212     };
213 };
214
215 #endif
216

```